Victorian 6502 User Group Newsletter

KAOS

For People Who Have Got Smart

OSI SYM KIM AIM UK101 RABBLE 65

Registered by Australia Post
Publication No. VBG4212

ADDRESS ALL CORRESPONDENCE TO 10 FORBES ST., ESSENDON, VIC. 3040

Vol.3 No.11 August 1983

It has finally happened. Jeff Rae is in his new house and with luck will have the same phone number for more than a couple of months. It was getting to be quite a job just changing the heading, speaking of headings, the new one has all the up to date phone numbers listed and you will notice we have dropped the APPLE and ORANGE and added the RABBLE 65.

After a false start last month (we hope not too many newsletters were destroyed trying to remove the sticky label) we have now decided on the new wrapper. Each month, KAOS will arrive hygenically sealed (untouched by human hands) in a polythene bag.

The reason for the change was that the way we were printing the wrappers was time consuming and very hard on David Anear's printer. We originally planned to put the stamp etc. on the back page, fold it in half and close it with a staple using sticky labels for the address. We were all set up to do this when Australia Post struck again - we got a letter informing us that as from January 1984 the largest opening allowed in a registered publication would be 88mm. This makes sense as it prevent letters getting caught inside newsletters etc. so it was back to the drawing board and the most practical solution was to use plastic bags. We would appreciate any information on damaged bags, if there are any problems we will change to a heavier plastic.

The next meeting will be on Sunday 28th August at 2pm at the Essendon Primary School, which is on the corner of Raleigh and Nicholson Streets, Essendon. Please note that the school will be open at 1pm.

The closing date for articles for the September newsletter is the 9th September.

INDEX

Alpha and I 14
Character Sets 7
Comp-Dos 1.3 Bug 10
Dumb Terminal Routine 6
For Sale 15
Give Away 3
Handicapping 4
I/O Machine Code 11
Machine Language pt 14 2
Meeting - KAOS 9
Meeting - Sydney 15
Meeting - W.A. 15
Modem Board 15
My Superboard pt 5 8
Software Review 5
SUPERBOARD 4
T.A.B. Letter 10
Tasan Board Mod 9

THE BEGINNING MACHINE LANGUAGE PROGRAMMER .....Part 14

by David Dodds

This month BMLP will tidy up a few loose ends and then look at the keyboard debouncing routine (DBOUNC) from ARTIST.

Back in part eleven when the ADC and SBC opcodes were covered, I mentioned the need to set and clear the carry flag for multiple precision work. The instructions SET and CLEAR are used to condition the status registers to a known state. SET is used when the register is required to be 'true' (1), and CLEAR when the value should be 'false' or 0. The CLEAR operation may be applied to the Carry (C), Overflow (V), Decimal mode (D) and Interupt (I) registers while SET applies only to C, D and I. The mnemonics for these operations consist of the first 2 letters of the instruction and the register name ie

```
SEC set the carry
CLC clear the carry
CLV clear the overflow
SED set the decimal mode
CLD clear the decimal mode
SEI set the interrupt
CLI clear the interrupt.
```

DBOUNC uses CLC to indicate that an error has occurred. The routine will also introduce the use of the stack for temporary storage.

The stack may be accessed with the instructions PUSH and PULL. These were described back in part 2 of the series. PUSH and PULL may only be used with two registers -the Accumulator and the Status register (P). The mnemonics for the instructions are :-

```
PHA Push Accumulator
PHP Push status register
PLA Pull accumulator
PLP Pull status register.
```

The technique used for debouncing is to program a delay of approximately 5 milliseconds, then to re-read the keyboard and compare the new value with what was previously obtained. At the completion of the routine the carry flag signals whether the keystroke has been verified.

At the start of DBOUNC the Accumulator contains the keyboard column value and the X register holds the index into the keyboard tables. Both these registers will be required during DBOUNC but the values they contain must be preserved. The first step is to store the accumulator at KYDOWN where it will be available for comparison after the re-read. The stack will be used to temporarily store the value of the X register so it is transferred to the accumulator which is then pushed to the stack. The first few lines of DBOUNC looks something like

```
DBOUNC STA KYDOWN
       TXA
       PHA
```

Now a call to the 5 millisecond delay routine DELAY5:

```
       JSR DELAY5
```

Next re-read the keyboard, convert it for C1 if necessary and compare the value to KYDOWN:

```
       LDA $DF00
       JSR CONVRT
       CMP KYDOWN.
```

If the two are equal then carry (C) and zero (Z) will both be set. The X register can therefore be restored to its original state before returning. If not equal then Z will be clear but C may be set or clear so it is necessary to ensure that it is clear before returning. The following sequence takes care of the necessary sorting:

```
        BNE DB1
        BCS DB2
DB1     CLC
DB2     PLA
        TAX
        RTS
```

While running the 6502 performs a cyclic operation of fetch the next instruction....execute it....
This fetch/execute operation is known as a machine cycle. The time taken for a machine cycle is directly related to the clock speed of the CPU. Each instruction on the 6502 requires a known number of machine cycles to complete. If the clock speed is known it is possible to calculate almost exactly how long a given sequence of instructions will take to execute so that a delay of any duration can be programmed.

Taking a simple case of

```
DELAY   DEX
        BNE DELAY
```

The DEX requires 2 machine cycles and the BNE takes 3. At 1MHz this loop will take 5 microseconds for each pass of the loop. Assuming that X was 0 when the loop was entered then 1280 (256*5) microseconds or 1.28 milliseconds will elapse before the loop is completed.

Longer delays can be made by adding another loop eg

```
DELAY   DEX
        BNE DELAY
        DEY
        BNE DELAY
```

By setting the X and Y registers to appropriate values the desired 5 millisecond delay can easily be achieved. Incrementing or decrementing memory locations could also be used to achieve the same result.

The subroutine DELAY5 in ARTIST is:

```
DELAY5  LDX #$C8
        LDY #5
        JSR DELAY
        RTS

DELAY   DEX
        BNE DELAY
        DEY
        BNE DELAY
        RTS
```

As an exercise see if you can calculate the exact time of the delay. The opcode matrix of the 65C02 published a few months back shows the number of machine cycles for each instruction. Don't forget to count in the LDX, LDY, JSR and RTS instructions!

GIVE AWAY

ABC lissajous patterns in blue, suitable for ironing onto T-shirts. The ABC no longer uses this pattern. Limit of 4 with instructions per member. The kids will love em. Send a S.A.E. to OSUG (address on page 4) with the number you require. White pattern on a blue background, or a blue pattern on a clear background.

Superboard
August 1983.
Newsletter of the Ohio Superboard User Group, 146 York Street, Nundah, 4012.

HANDICAPS TO ODDS

The simulated computer tote program last month may have given you a clue to my favourite pastime (some would say obcession), horse racing. With a 17% take from the pool, it is extremely difficult to win consistently at racing. My guess is that fewer than 1 in every hundred punters win, and I like challenges! Computing was a challenge that I took up in 1980, and I have used the computer to do some of the repetitive, boring tasks.

The selection of winners is called handicapping, and involves two basic procedures.

The first is done after a race meeting. You consider weight carried, rider, beaten margin, class of race, quality of horses running, and any mishaps that may have occurred in the race to each horse, and express all of the above in a figure that represents the effort of that horse in the race. You do this for all the starters in every race. The figure is called a rating.

The second is the handicapping of the fields of horses before they race. You look at the ratings of the last few runs of each contender, and select the one which was given in the race which most closely resembles the race to come. You then apply corrections for the barrier position, weight to be carried, rider, suitability of distance and track, ability to handle the going, improvement expected after a spell, and whatever else you think is valid. After you have done this for each horse running, you have your handicap. The task now is to estimate the chance of each horse in each race, and convert the chance to odds.

On the racecourse, you only bet on your selections if the odds you can obtain from bookmaker or tote is <u>better</u> than your estimated odds. In some races, two or even four of your selections might be a bet. Over a long term, if your handicapping remains accurate, you will win.

Handicaps to odds: Over a couple of years, you keep records of how your handicaps have performed. You might find that one of your top six horses wins 85% of your races on average. Delving further, you find that for every 10 horses that win on a Ø handicap, 8 win on a 1 Kilogram handicap, 5 win on a 2.5 Kg handicap, 2 on a 4.5 Kg handicap, and so on. The program below uses these long term probabilities to estimate the chances in future races. Please note that it is <u>important</u> that the probabilities <u>do</u> reflect <u>your own</u> handicaps. This means that the figures that I have used may be worthless to you. I only use them to demonstrate the program.

Variables used:-
- L is the lowest handicap.
- CH is the probability that one of your selections will win.
- T total of probabilities, needed to calculate the odds.
- P$ allows you to change CH if you think this race is easier or harder than average to handicap.
- R$ is State and race number, used in the printout.
- A$(R) is horses names or numbers used in the printout.
- N is the number of chances in the race.
- H(R) is several things associated with each horse.
- D is the data accumulated from past results, in ½ Kg steps. This should be changed to reflect your own success rate, as should CH in line 45.

```
1Ø DIMN$(2Ø),H(2Ø),P(2Ø):FORD+ØTO19:READP(D):NEXT
15 PRINTCHR$(26);TAB(3);"HANDICAPS TO ODDS":L=99:T=Ø:CH=8Ø
2Ø PRINT:PRINT:INPUT"State & Race";R$:PRINT
25 INPUT"How many chances";N
3Ø FORR=1TON:PRINTCHR$(26):IFR>1THENPRINT"Next ";
35 PRINT"Horse's name":INPUTN$(R):PRINT
4Ø INPUT"Final Handicap";H(R):IFH(R)<LTHENL=H(R)
45 NEXT:FORR=1TON:D=INT((H(R)-L)*2):IFD>19THEND=19
5Ø H(R)=P(D):T=T+P(D):NEXT
55 PRINTCHR$(26);R$;TAB(13)CH"% odds":CH=CH/1ØØ:PRINT
6Ø FORR=1TON:PRINTN$(R);TAB(16)INT(T/(H(R)*CH)*1Ø+.5)/1Ø-1
65 IFN<1ØTHENPRINT
7Ø NEXT:PRINT:INPUT"Different %";CH$:IFCH$="N"GOTO15
75 CH=VAL(CH$):GOTO55
8Ø DATA 1,.9,.8,.67,.57,.5,.4,.33,.25,.2,.14
85 DATA .11,.Ø8,.Ø67,.Ø5,.Ø4,.Ø33,.Ø25,.Ø17,.Ø1
```

SOFTWARE REVIEW - Interceptor

Interceptor is an Aardvark M/C arcade game occupying $0DE5 to $1FFF. The object of the game is to use your interceptor to protect your cities from an endless horde of kamakazi invaders, which dodge in to crash on them. Your craft has limitless ammunition, and if hit by an invader before you can shoot it, another appears, after a delay, in the launching platform. During this delay, of course, the invaders keep coming. Once you have left the launch pad, you can manoevre in any direction, and hover, and your aircraft has inertia, making flying more difficult, but realistic.

To assist your defence, you have two indestructible computer fired cannon set up on hills on either side of the city. The firing angles are fixed. Also, friendly fire can do you no harm. The cannon will target you, but cannot shoot you, and any wild firing on your part cannot harm the cities.

The automatic cannon give the game an interesting feature normally seen only on dedicated arcade machines, the object being to attract attention. Left on their own to defend the cities, the cannon and invaders will play a preprogrammed game and score 6560 points, interesting to watch.

The score is continually updated on the top left of your screen, and after each attack wave, you get a rest while bonus points are added. The speed of the invaders also increases, and above 20,000 points, is so fast that skill takes second place to pure reflexes. You get a bonus city every 11,000 points, and the game ends when all cities are destroyed.

And now for my pet grievance! This is the third Aardvark M/C game I have reviewed where no skill is required to get a respectable score. By staying on the launch pad, and firing continuously straight up, I scored 20,000 and 24,000 points in two attempts. It took dozens of games when I used all the controls, before I was able to beat those scores, and then I did it once only, with 26,270 points. To cure this problem, firing should not be possible until your interceptor has left the protection of the launching platform.

Two keys I found difficult to use were 8 (thrust) and Space Bar (fire). These have to be operated by forefinger and thumb, which was tiring on the arm.

To sum up:- Not a bad game, with plenty of action, and a combination of skill and reflexes needed to score well. A 24 x 24 version is available for the perusal of library members at the usual postage rates.

— SUPERBOARD —

DUMB TERMINAL ROUTINE FOR THE AUSTRALIAN BEGINNING

Using the information given in the July KAOS newsletter, here is a routine which will turn your computer into a terminal, for the purpose of communications with the Australian Beginning.

```
01AD A9 03     LDA #$03      Set up ACIA for receive interrupts, 8
01AF 8D 00 F0  STA $F000     data bits, no parity, 1 stop bit.
01B2 A9 95     LDA #$95      Note that C4 owners with Synmon monitor
01B4 8D 00 F0  STA $F000     should use $FC00
01B7 58        CLI           Allow interrupts to microprocessor.
01B8 20 00 FD  JSR $FD00     Get a character from the keyboard.
01BB 20 B1 FC  JSR $FCB1     Send to ACIA. Note for C4 use $BF15
01BE D0 F8     BNE $01B8     Branch always to get another character.
01C0 48        PHA           Come here on interrupt.
01C1 AD 01 F0  LDA $F001     Get the character received in the ACIA.
01C4 30 03     BMI $01C9     Ignore it if not ascii.
01C6 20 2D BF  JSR $BF2D     Print it on screen.
01C9 68        PLA           See text.
01CA 40        RTI           Go back to whatever micro was doing.
```

Why in the stack area? Well, that's where OSI in their wisdom put the IRQ vector; at $01C0. The program is designed for a full duplex system, where there is simultaneous transmission and reception of data, and what you send is echoed back to you, when it prints on the screen.

HARDWARE MODIFICATION

Since OSI use a halting Get-Key routine at $FD00, we need some means of informing the microprocessor whenever a character has been received from the mainframe. This job was made for Interrupts. To option this facility, unused by OSI, we need to solder a small jumper lead from pin 7 on the ACIA (6850) to pin 4 on the Microprocessor (6502).

This should cause no problems when the computer is in use with other programs but if you are uneasy about it, then wire in a switch.

OPERATION

You enter the routine at $01AD. Once it reaches $01B8, it sends a character to the ACIA every time you press a key. Since you can't get CHR$(0) from the keyboard, the BNE instruction becomes branch always, back to get another key.
When a full character is received in the ACIA from the mainframe, the ACIA causes an IRQ interrupt. The microprocessor then finishes whatever instruction it was doing, stores the status register and program counter on the stack and jumps to our routine at $01C0. First, we store the accumulator contents. Normally, we'ed also store the X and Y registers, however this is already done in the print routine at $BF2D, so it saves us the trouble. We don't need to check the status of the ACIA, since it has already indicated it is ready by sending the interrupt, so we can get the character. I presume that TAB might use the 8th bit to send control characters, and we don't want Tanks and Guns on the screen, so we bypass the print routine if the 8th bit is used. Otherwise, we print the character, get back the accumulator, and return control to whatever part of the program the microprocessor was executing before it was interrupted by a message from TAB.

Hope it works,

Ed Richardson.

ADDING IMPROVED CHARACTER SETS

by Jeff Kerry

Here is a simple way to increase the usefulness and readability of your video screen. New character generator ROM chips are available from Bernie Wills in Queensland, and they can be quite simple to install.

STANDARD SET: The usual 256 characters consisting of the ASCII alphanumerics and graphic and game characters.

ENHANCED SET: Reshaped alphanumerics, with proper descenders on g's and y's etc.; two complete chess sets in white and black; the "missing" 3/4 blocks, and new game and effect shapes, replacing rarely used originals.

SCIENTIFIC SET: Contains the enhanced alphanumerics and graphics, but with game characters replaced by: subscripts and superscripts; a Greek alphabet; and an array of scientific and mathematical characters including square root, not-equal, proper divide-by, angle symbols, etc.

HI-RES GRAPHICS SET: 256 small block elements based on 2-pixel by 4-pixel blocks, giving 128 x 128 resolution on a 64x32 screen, or 48 x 96 on a 24x24 screen.

The character sets are available in standard 2716 ROMs or, more usefully, in 2732 ROMs which provide you with two full character sets in the one chip. By piggybacking these you can get all four in one chip socket location. You can enable different sets with a simple switch arrangement, or use a software latch so a program can select any set on the run. Unfortunately you can't mix them on the screen at the same time.

Bernie supplies plenty of documentation about the chips, including some instructions about installing them. However if, like me, you have a Tasan video board, there is an easy way to install all four character sets if you buy them as two 2732's.

The Tasan board video latch at address 56900 has four lines, two of which are used for selecting 64 or 32 screen width, and normal or inverse video. I found the two spare lines to be ideal for selecting the new ROM character sets. This means that you can simply POKE the latch with a number to select ANY combination of any character set in any screen format! This retains complete compatibility with the original format (my usual policy).

Installation doesn't even require any track cutting, just a few minutes skilful soldering and three bits of wire. CAREFULLY solder the two 2732's together in piggyback formation, EXCEPT for the pin 18's - bend these out to the side. Bend out the pin 21 of the LOWER chip, and insert the assembly in the original character generator ROM's socket, except for pins 18 and 21.

Now locate chip U51, a 74175. Connect the joined pins 21 of the ROMs to pin 10 of the underside of U51 socket, and the two pin 18's of the ROMs to pins 6 and 7 of U51. The video board will "wake up" on power-up or reset, using the character set in the "top half" of whichever ROM you connect to pin 7 of U51. Poking numbers into 56900 will select combinations of character sets and screen formats.

COST: 2716 $12-80 2732 $17-80
(State which character sets you want in each chip)

Enquiries to:
B. Wills

MY SUPERBOARD Series II. Part 5
by John Whitehead

The Tasker Bus system. The start of my expansion.

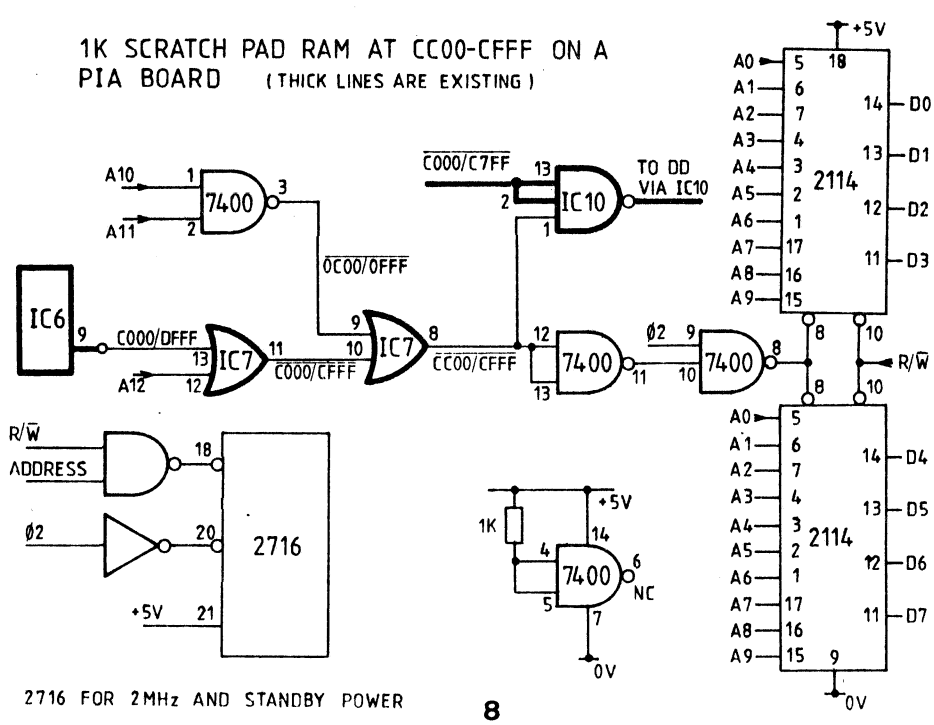
All of my expansion is with the Tasker bus and if this is not connected, my Superboard works normally as an 8k system.

The advantage of the Tasker bus is that individual cards can be removed for modification etc. without loosing the whole system. The disadvantage is that there are 5 plugs and sockets between the Superboard and any one card and that each card requires its own decoding chips. The only problem in practice is that the cards are not gold plated at the edge connector and need cleaning when they fail to work correctly. I am presently trying smearing the card connector with silicon grease to prevent it oxidising.

I started out with a PIA/VIA board without a VIA, and an EPROM board. The EPROM board I wired up for 2128 2k x 8 TTL RAM chips so that I could develop programs in RAM at the final address, burn them into EPROM and replace the RAM with a 2716 EPROM. I built a 2716 EPROM programer from the circuit in KAOS Oct. 81 to connect to the PIA and wrote software for it in BASIC.

My first EPROM contained the original Extended monitor at $E800 to $EFFF. My next programs to put in EPROM were the teletype routine and Real time clock but for this I needed a few bytes of scratch pad RAM. I didn't like using any existing RAM in case it conflicted with other programs. To overcome this I put two 2114 RAMs and a 7400 on veroboard and wired it up onto the PIA board as shown in the circuit below. As the PIA is decoded at $C004 and 5, it was easy to decode the address lines to put the RAM at $CC00 to $CFFF. (The RABBLE has scratch pad ram at $C800 to $CFFF). So far I have reserved $CFEA to $CFEF for teletype, $CFF0 to $CFF9 for Real time clock, $CFFA to $CFFC for NMI and $CFFD to $CFFF for IRQ.

All the above worked well at 1MHz but would not work at 2MHz. I found that 2716 EPROMs have two chip select pins and one is nearly four times faster than the other. Outputs are valid 450ns after pin 18 goes low and 120ns after pin 20 goes low. As R/W signal and address decoding go low first, these are connected to pin 18 and as 02 goes low later this can be connected to pin 20. When connected like this 2716s will work at 2MHz and when not being accessed, draw only 25% of normal power.



2716 FOR 2MHz AND STANDBY POWER

YET ANOTHER TASAN VIDEO BOARD MOD

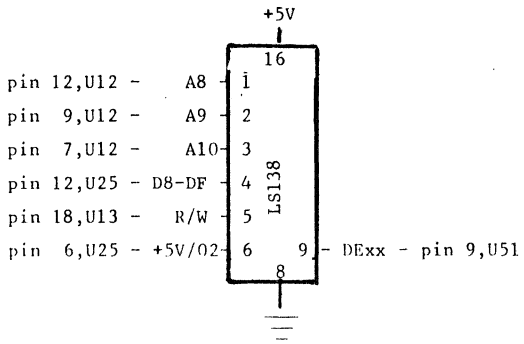
by Paul Dodd

If you have tried to run any standard C4P or C8P disk software on the Tasan Video Board, you may notice that the screen sometimes changes to inverse video or 32X32 mode (try running 'Plot Basic' and see the effects!).

The problem is that the video latch on the video board is located at $D800 - $DEFF (or something like that). The standard video latch is at $DE00-$DEFF.

To modify the video board, piggy-back a 74LS138 on to U25 (which is also a 74LS138).

The new circuit looks like this:



Solder pins 6, 8 and 16 to the chip below.
With a short piece of wire, solder pin 12 of U25 to pin 4 of the chip above.
Solder a piece of wire from pin 1 of the piggy-backed chip to pin 12, U12 (74LS241).
Solder a wire from pin 2 of the piggy-backed chip to pin 9, U12.
Solder a piece of wire wrap wire from pin 3 of the new chip to pin 7, U12.
Solder a piece of wire from pin 5 of the piggy-backed chip to pin 18, U13 (74LS241).
Cut the track to pin 9, U51 (74LS175).
Lastly join a wire from pin 9 of the piggy-backed chip to pin 9 of U51.

THE MEETING WAS KAOS

by King Corky

This months meeting almost turned into a vendors gathering, with various members, myself included, endeavouring to sell their wares. Bill Chilcott gave us a run down on the latest from the labs at the Rabble den, viz.. the 2nd batch of Rabble 65 boards are on sale, with one minor artwork bug which will be fixed for the 3rd batch. Prototype cases are also available for the main board and the keyboard but the keyboards are not yet available.

George, father of the yewbewt machine, held in the palm of his, tiny?, hand a switch-mode power supply capable of +-5, +-12 volts at some ridiculous current level, in a case that looks small enough to be a toy train battery pack. Cost? $100. George also mentioned that his new Assembler is running, but for the Rabble 65, 64 screen/disk version only, for $65. David Wilson held up a promo blurb on the new Rockwell 6502 lookalike chip that sports two (2) uP on the one chip, plus I/O & RAM, plus larger registers. The dual processors share the same memory addresses yet can run independantly, I think.

MICRO on the OSI was on show and Ron Kerry will be doing a review on it shortly.

The Rabble Bored must be trying to keep a low profile, they were relegated to the back benches, again. Life member of the Bored, Michael Lemaire, (I don't know if this is for public knowledge yet or not, but in the best interests of the public and in the spirit of our politicians..), Michael is developing the 'front-end' routine for the TAB, which will give all those with modems and OSI's, access to the TAB plus downloading facilities. David Dodds has available copies of the complete source code of his Artist routine that has been running in the newsletter as an instructional series, for free. Peter Carless was showing off his C1/C8 twin 8" disk system (very nice thank you) which is the same system he is selling for $1890. A steal when you include the software.

COMPDOS 1.3 BUGS

by Michael Lemaire

Alas, there are a couple of bugs (features?) in the renowned all-singing, all-dancing Compdos version 1.3; they won't trouble you, though, unless you find them (philosphical point, that).

First: the drive selection on commands (eg. dr%b) is not very choosy about what follows the '%' character. If, for instance you type 'dr%v' when you really meant 'dr%c', then you can say goodbye to the directory track of the current drive. The bug has been located, and Paul will be wearing sackcloth and ashes for the next two weeks.

Scond: the zero command should be invoked with a file name, NOT with a track range. As written, it calls a routine in 65D which will result in the zeroing of every track from the specified start track to the end of the disk. Moral: use file names (we can justify this bug/feature on the grounds of hardware independent programming, besides it's too much effort to change).

If you have a (legal) copy of Compdos 1.3, you can bring it into COMP-SOFT to get an update for the first bug mentioned above.

THE AUSTRALIAN BEGINNING NEWS

Unfortunately, the deadline for this newsletter is a few days too soon for us! The news that we promised you in the last newsletter can not be released until late next week. However, to make up for this, we will give you full details in a special release in the next newsletter!

I would like to make it quite clear to all KAOS members that we are now no longer (in any way) associated with the Computer Country group of companies or management thereof!

The Australian Beginning is alive and very well and will continue to grow from strength to strength, especially if we continue to get valuable feedback from our many members.

Noel Fenton
Chief Executive Officer
The Australian Beginning

Hi there! I still haven't heard from many KAOS members re the $5 credit. You've got one week folks - I go on holidays to sunny Bali on the 22nd August for 2 glorious weeks, so get in touch soon. While I'm away, Andrew Lighten will be standing (sitting?) in for me and will help you with most queries you may have, but some may have to wait until I get back on the 4th September (give me a couple of days to catch up on my mail!).

Any KAOS members who belong to other clubs may be interested to know that we have a special deal for club - contact me by phone (03 813 1133) or by electronic mail (username : Davidlutz) for further details - it's well worth it!

David Lutz
Customer services manager
The Australian Beginning

MACHINE CODE I/O MODULE

by Michael Lemaire

A while ago, I was writing some assembler programs and found that I needed to handle disk file I/O. I also found that there is no facility for it on 65D. As a result, I wrote this module, which can handle I/O transfers for all 65D devices in a nice, clean manner.

The module contains three routines: GETCH, PUTCH and PUTSTR which handle character input, output and string output to or from any device. They are called in a reasonably consistent manner; X holds the device number, A holds a character (returned by GETCH, sent to PUTCH), and strings are null-terminated and pointed to by A (high) and Y (low).

To handle disk files, the routines BUFSET, BUFREL, FOPEN and FCLOSE exist. BUFSET and BUFREL set and release the buffers for devices 6 and 7 by sending command strings to the DOS via the DOSCMD routine; this is a convenient routine for any DOS command. FOPEN and FCLOSE are reasonably well documented; the FILEVAR table indexing may need further explanation. FILEVAR is a block of RAM in the I/O distributor of 65 which holds information on the current track number, the maximum track number, the buffer location and so on for the devices 6 and 7. There are 8 bytes for device 6, followed by 8 bytes for device 7, hence FILEVAR is usually accessed with an offset of 0 for device 6 and an offset of 8 for device 7.

There are a couple of extra routines called by FOPEN -- these are SETDV6 and SETDV7. These set up the buffer pointers for the appropriate device, and also fix a bug (feature?) in 65D.

When I first tried running the module, it crashed whenever a buffer fault (ie. read or write past the "end" of the track in the buffer) occurred. No bang, nor even a whimper; it just dropped in its tracks. I finally traced the problem to an area in the I/O distributor where the buffer fault test takes place. According to the code, if a buffer fault occurs, instead of doing something silly, like jumping to the buffer fault handler, it jumps into the middle of the directory buffer--I suppose it takes all sorts..

I suspect that this has something to do with the BASIC get/put overlays; if someone knows I would greatly appreciate the information.

There are three variables in the module called STDIN, STDOUT, and STDERR. I've copied the names from Unix because (1) I like Unix, (2) I like the names, and (3) I couldn't be bothered inventing new names. The idea of these variables is to store device numbers for normal input, output, and runtime errors. This allows a program to be changed, or have its output, say, redirected merely by changing the contents of these variables, and using eg. LDX STDOUT; JSR OUTCH to send output to the "normal" output device. Yes, they have a similar function under Unix.

The module may not appear to be the most efficient thing ever written; this is because the second law of software engineering states that efficiency is th root of all evil. The module was thus written with efficiency considerations being generally ignored in favour of nice portable constructs. For instance, the buffer-set and buffer-release routines could be written to call the COMPDOS routines directly; but this would result in needing to modify the module for every different memory size or version of COMPDOS. The method used in the module, while slower, will always work if COMPDOS is there in its little RAM universe, in any shape or form.

```
OS65D MACHINE CODE FILE HANDLER MODULE    PAGE # 1

LOC  OBJ        SEQ  SOURCE
0000            3    ;          OPT OBJECT
0000            4    ; I/O MODULE
0000            5    ;===========
0000            6    ; I/O handler module for machine code.
0000            7    ; Runs under OS65D3.3 with Compdos 1.3.
0000            8    ;       ajl
0000            9    ;   Last update 10/8/83
0000            10   ;
0000            11   VECT      EQU $FC  General-purpose (ind),y vector.
0000            12   ; External references..
0000            13   STROUT    EQU $2D73
0000            14   SETTK     EQU $26BC
0000            15   CALLTK    EQU $2B1A   Call in sector.
0000            16   SAVETK    EQU $2C37   Save sector.
0000            17   SECTNM    EQU $265E
0000            18   MEMVECT   EQU $FE
0000            19   BUFVECT   EQU $E1   Ptr to DOS input buffer.
0000            20   DOSBUFPT  EQU $2CE5  DOS input buffer ptr.
0000            21   DOSINTRP  EQU $2A84 DOS command interpreter.
0000            22   FILEVAR   EQU $2326  DOS disk file vars.
0000            23   FINDFILE  EQU $2DA6  Find file in dir; name in input buffer.
0000            24   INPTAB    EQU $2301  I/O distributor device input vects
0000            25   OUTTAB    EQU $2311  Same for output.
0000            26   IOCHR     EQU $2363  DOS (sometimes) uses this for I/O param.
0000            27   ;
6000            28             ORG $6000
6000 4C 87 61   29             JMP MAIN
6003            30   ;
6003            31   ; standard I/O streams (dev #'s)..
6003 02         32   STDIN     BYTE 2
6004 02         33   STDOUT    BYTE 2
6005 02         34   STDERR    BYTE 2
6006            35   ;
6006 00         36   DEVTEMP   BYTE 0
6007            37   ;
6007            38   ; DOS.CMD--Send command string to DOS interpreter.
6007            39   ;  Call with pointer to null-terminated string
6007            40   ;  in A(hi) Y(lo).
6007 84 FC      41   DOSCMD    STY VECT
6009 85 FD      42             STA VECT+1
600B A0 00      43             LDY #0
600D B1 FC      44   DOSCMD1   LDA (VECT),Y
600F F0 05      45             BEQ DOSCMD2
6011 91 E1      46             STA (BUFVECT),Y
6013 C8         47             INY
6014 D0 F7      48             BNE DOSCMD1 (BRA)
6016 A9 0D      49   DOSCMD2   LDA #13  CR.
6018 91 E1      50             STA (BUFVECT),Y
601A 20 84 2A   51             JSR DOSINTRP
601D 60         52             RTS
601E            53   ;
601E            54   ;The following equates are used to fool Compdos
601E            55   ;into believing that BASIC is resident in memory;
601E            56   ;otherwise it will not set or release buffers..
601E            57   CDKLUDGE  EQU $2C00  Test address.
601E            58   KLUDGDAT  EQU $00    Data expected for BASIC.
601E            59   ; (Actually anything but $2A will work)
601E            60   ;
601E            61   ; BUFSET--Set buffers for dev's 6 & 7..
601E AD 0D 2C   62   BUFSET    LDA CDKLUDGE   Save current flag value..
6021 48         63             PHA
6022 A9 00      64             LDA #KLUDGDAT  Conspire to defraud Compdos..
6024 8D 0D 2C   65             STA CDKLUDGE
6027 A9 60      66             LDA #BUFSET1/256
6029 A0 3A      67             LDY #BUFSET1
602B 20 07 60   68             JSR DOSCMD
602E A9 60      69             LDA #BUFSET2/256
6030 A0 40      70             LDY #BUFSET2
6032 20 07 60   71             JSR DOSCMD
6035 68         72             PLA          Restore kludge data.
6036 8D 0D 2C   73             STA CDKLUDGE
6039 60         74             RTS
603A 42 53 20   75   BUFSET1   BYTE 'BS 06',0
603D 30 36 00
6040 42 53 20   76   BUFSET2   BYTE 'BS 07',0
6043 30 37 00
6046            77   ;
6046            78   ; BUFREL--Release the buffers..
6046 AD 0D 2C   79   BUFREL    LDA CDKLUDGE
6049 48         80             PHA
604A A9 00      81             LDA #KLUDGDAT
604C 8D 0D 2C   82             STA CDKLUDGE
604F A9 60      83             LDA #BUFREL1/256
6051 A0 5B      84             LDY #BUFREL1
6053 20 07 60   85             JSR DOSCMD
6056 68         86             PLA
6057 8D 0D 2C   87             STA CDKLUDGE
605A 60         88             RTS
605B 42 52 00   89   BUFREL1   BYTE 'BR',0
605E            90   ;
605E            91   ;
605E            92   ; FOPEN--Open (disk) file; name pointed to by
605E            93   ;  A(hi) Y(lo)--null-terminated string ofcourse.
605E            94   ;  Dev # (6 or 7) in X
605E 8E 06 60   95   FOPEN     STX DEVTEMP
6061 84 FC      96             STY VECT
6063 85 FD      97             STA VECT+1
6065            98   ; Copy file name to input buffer..
6065 A0 00      99             LDY #0
6067 B1 FC      100  FOPEN1    LDA (VECT),Y
6069 91 E1      101            STA (BUFVECT),Y
606B F0 03      102            BEQ FOPEN2
606D C8         103            INY
606E D0 F7      104            BNE FOPEN1  (BRA)
6070 8D E5 2C   105  FOPEN2    STA DOSBUFPT  (A==0)
6073 A9 0D      106            LDA #13   CR at end of name..
6075 91 E1      107            STA (BUFVECT),Y
6077 20 A6 2D   108            JSR FINDFILE
607A AC 06 60   109            LDY DEVTEMP   Use Y as index into FILEVAR table..
607D C0 06      110            CPY #6        Dev #6?
607F D0 04      111            BNE FOPEN3
6081 A0 00      112            LDY #0        If so, offset==0.
6083 F0 02      113            BEQ FOPEN4    (BRA)
6085 A0 08      114  FOPEN3    LDY #8    Offset for dev#7.
6087 99 2A 23   115  FOPEN4    STA FILEVAR+4,Y   A==start tk.
608A BD 7A 2E   116            LDA $2E7A,X   Get end tk.
608D 99 2B 23   117            STA FILEVAR+5,Y
6090 A9 00      118            LDA #0
6092 99 2D 23   119            STA FILEVAR+7,Y  Clear dirty flag.
6095            120  ; Set up buffer input/output pointers..
6095 C0 00      121            CPY #0    dev#6?
6097 D0 06      122            BNE FOPEN5
6099 20 C4 60   123            JSR SETDV6
609C 4C A2 60   124            JMP FOPEN6
609F 20 E7 60   125  FOPEN5    JSR SETDV7
60A2            126  FOPEN6    EQU *
60A2            127  ; Call in first track..
60A2 B9 2A 23   128            LDA FILEVAR+4,Y
60A5 99 2C 23   129            STA FILEVAR+6,Y  Set "current" tk
60A8 8C 06 60   130            STY DEVTEMP
60AB 20 BC 26   131            JSR SETTK
60AE AC 06 60   132            ldy DEVTEMP
60B1 A9 01      133            LDA #1        Sect#
60B3 8D 5E 26   134            STA SECTNM
60B6 B9 26 23   135            LDA FILEVAR,Y   Buffer addr lo
60B9 85 FE      136            STA MEMVECT
60BB B9 27 23   137            LDA FILEVAR+1,Y Buffer addr hi
60BE 85 FF      138            STA MEMVECT+1
60C0            139  YYYY      EQU *
60C0 20 1A 2B   140            JSR CALLTK  Go get tk
60C3 60         141            RTS
60C4            142  ;
60C4            143  ; I/O pointer setting for dev's 6&7..
60C4            144  ; At the same time, fix the buffer-fault vectors
60C4            145  ; to avoid little nuisances ie. crashes.
60C4 AD 26 23   146  SETDV6    LDA FILEVAR    Buff start lo
60C7 8D AC 23   147            STA $23AC    INPTR
60CA 8D C3 23   148            STA $23C3    OUTPTR
60CD AD 27 23   149            LDA FILEVAR+1   Buff start hi
60D0 8D AD 23   150            STA $23AD
60D3 8D C4 23   151            STA $23C4
60D6 A9 CC      152            LDA #$CC    Buff fault lo byte
60D8 8D A9 23   153            STA $23A9
60DB 8D BF 23   154            STA $23BF
60DE A9 23      155            LDA #$23
60E0 8D AA 23   156            STA $23AA
60E3 8D C0 23   157            STA $23C0
60E6 60         158            RTS
60E7            159  ;
60E7 AD 2E 23   160  SETDV7    LDA FILEVAR+8   Buff start lo
60EA 8D FD 23   161            STA $23FD    INPTR
60ED 8D 16 24   162            STA $2416    OUTPTR
60F0 AD 2F 23   163            LDA FILEVAR+9
60F3 8D FE 23   164            STA $23FE
60F6 8D 17 24   165            STA $2417
60F9 A9 20      166            LDA #$20  Buff fault vect lo byte
60FB 8D FA 23   167            STA $23FA
60FE 8D 12 24   168            STA $2412
6101 A9 24      169            LDA #$24   hi byte
6103 8D FB 23   170            STA $23FB
6106 8D 13 24   171            STA $2413
6109 60         172            RTS
610A            173  ;
610A            174  ; FCLOSE--Close file; flush if necessary.
610A            175  ; Call with dev# (6 or 7) in X
610A E0 06      176  FCLOSE    CPX #6
610C D0 04      177            BNE FCLOSE2
610E A0 00      178            LDY #0    Index into FILEVAR table.
6110 F0 06      179            BEQ FCLOSE4  (bra)
6112 E0 07      180  FCLOSE2   CPX #7
6114 D0 34      181            BNE FCLOSE6  is not file device; just ignore.
```

```
6116 A0 08      182           LDY #8    Index.
6118 B9 2D 23   183 FCLOSE4   LDA FILEVAR+7,Y  Buffer dirty?
611B F0 2D      184           BEQ FCLOSE6   No; return.
611D            185 ; Flush buffer..
611D 8C 06 60   186           STY DEVTEMP
6120 B9 2C 23   187           LDA FILEVAR+6,Y
6123 20 BC 26   188           JSR SETTK
6126 AC 06 60   189           LDY DEVTEMP
6129 A9 01      190           LDA #1
612B 8D 5E 26   191           STA SECTNM
612E B9 26 23   192           LDA FILEVAR,Y  Buffer addr..
6131 85 FE      193           STA MEMVECT
6133 B9 27 23   194           LDA FILEVAR+1,Y
6136 85 FF      195           STA MEMVECT+1
6138 B9 29 23   196           LDA FILEVAR+3,Y  Calc. #pages..
613B 38         197           SEC
613C F9 27 23   198           SBC FILEVAR+1,Y
613F 20 37 2C   199           JSR SAVETK
6142 A9 00      200           LDA #0
6144 AC 06 60   201           LDY DEVTEMP
6147 99 2D 23   202           STA FILEVAR+7,Y  Clear dirty flag.
614A 60         203 FCLOSE6   RTS
614B            204 ;
614B            205 ; GETCH--Get char from input -) A.
614B            206 ;  Call with dev# in X.
614B CA         207 GETCH     DEX    [1..7]-)[0..6]
614C 8A         208           TXA
614D 0A         209           ASL A   Double for index.
614E AA         210           TAX
614F BD 02 23   211           LDA INPTAB+1,X  Push hi byte of addr..
6152 48         212           PHA
6153 BD 01 23   213           LDA INPTAB,X  Then lo byte
6156 48         214           PHA
6157 60         215           RTS    Jump to routine.
6158            216 ;
6158            217 ; PUTCH--Output char in A -) dev #(X)..
6158            218 ;
6158 8D 63 23   219 PUTCH     sta IOCHR   Save char.
615B CA         220           DEX    Fiddle for index as for GETCH..
615C 8A         221           TXA
615D 0A         222           ASL A
615E AA         223           TAX
615F BD 12 23   224           LDA OUTTAB+1,X
6162 48         225           PHA
6163 BD 11 23   226           LDA OUTTAB,X
6166 48         227           PHA
6167 AD 63 23   228           lda IOCHR   Get char back
616A 60         229           RTS
616B            230 ;
616B            231 ; PUTSTR--output null-terminated string.
616B            232 ; Call with pointer to string in A(hi) Y(lo),
616B            233 ; and dev# in X. Terminating null is not output.
616B 84 FC      234 PUTSTR    STY VECT
616D 85 FD      235           STA VECT+1
616F 8E 06 60   236           STX DEVTEMP
6172 A0 00      237 PUTSTR1   LDY #0
6174 B1 FC      238           LDA (VECT),Y
6176 F0 0E      239           BEQ PUTSTR2  End at null.
6178 AE 06 60   240           LDX DEVTEMP
617B 20 58 61   241           JSR PUTCH
617E E6 FC      242           INC VECT    This routine handles strings
6180 D0 F0      243           BNE PUTSTR1 of length )255 characters.
6182 E6 FD      244           INC VECT+1
6184 D0 EC      245           BNE PUTSTR1 (bra)
6186 60         246 PUTSTR2   RTS
6187            247 ;
6187            248 ; A useful macro for string output..
6187            249 MACRO     PRINTSTR P1,P2   P1==Dev#, char *P2.
6187            250           LDX P1
6187            251           LDA #P2/256
6187            252           LDY #P2
6187            253           JSR PUTSTR
6187            254           ENDM
6187            255 ;



6187            256 ; Test program..
6187            257 MAIN      PRINTSTR STDOUT,MESS1
6187 AE 04 60   +1            LDX STDOUT
618A A9 61      +1            LDA #MESS1/256
618C A0 CA      +1            LDY #MESS1
618E 20 6B 61   +1            JSR PUTSTR
6191 20 1E 60   258           JSR BUFSET
6194            259           PRINTSTR STDOUT,MESS2
6194 AE 04 60   +1            LDX STDOUT
6197 A9 61      +1            LDA #MESS2/256
6199 A0 D9      +1            LDY #MESS2
619B 20 6B 61   +1            JSR PUTSTR
619E            260           PRINTSTR STDOUT,FILENAME
619E AE 04 60   +1            LDX STDOUT
61A1 A9 61      +1            LDA #FILENAME/256
61A3 A0 FD      +1            LDY #FILENAME
61A5 20 6B 61   +1            JSR PUTSTR
61A8 A9 61      261           LDA #FILENAME/256
61AA A0 FD      262           LDY #FILENAME
61AC A2 07      263           LDX #7
61AE 20 5E 60   264           JSR FOPEN
61B1            265           PRINTSTR STDOUT,MESS3
61B1 AE 04 60   +1            LDX STDOUT
61B4 A9 62      +1            LDA #MESS3/256
61B6 A0 06      +1            LDY #MESS3
61B8 20 6B 61   +1            JSR PUTSTR
61BB A2 07      266 MAIN1     LDX #7
61BD 20 4B 61   267           JSR GETCH
61C0 AE 04 60   268           LDX STDOUT
61C3 20 58 61   269           JSR PUTCH
61C6 4C BB 61   270           JMP MAIN1
61C9 60         271           RTS
61CA 48 65 6C   272 MESS1     BYTE 'Hello there!',13,10,0
61CD 6C 6F 20
61D0 74 68 65
61D3 72 65 21
61D6 0D 0A 00
61D9 42 75 66   273 MESS2     BYTE 'Buffers are set.',13,10
61DC 66 65 72
61DF 73 20 61
61E2 72 65 20
61E5 73 65 74
61E8 2E 0D 0A
61EB 4E 6F 77   274           BYTE 'Now opening file:',0
61EE 20 6F 70
61F1 65 6E 69
61F4 6E 67 20
61F7 66 69 6C
61FA 65 3A 00
61FD 54 45 58   275 FILENAME  BYTE 'TEXTFILE',0
6200 54 46 49
6203 4C 45 00
6206 52 65 61   276 MESS3     BYTE 'Reading from file...',13,10,0
6209 64 69 6E
620C 67 20 66
620F 72 6F 6D
6212 20 66 69
6215 6C 65 2E
6218 2E 2E 0D
621B 0A 00
621D            277           END

USER SYMBOLS   ( NUMERIC ORDER )
SYMBOL  VALUE
KLUDGDAT 0000 BUFVECT  00E1 VECT    00FC MEMVECT  00FE
INPTAB   2301 OUTTAB   2311 FILEVAR 2326 IOCHR    2363
SECTNM   265E SETTK    26BC DOSINTRP 2A8A CALLTK  2B1A
CDKLUDGE 2C0D SAVETK   2C37 DOSBUFPT 2CE5 STROUT  2D73
FINDFILE 2DA6 STDIN    6003 STDOUT   6004 STDERR  6005
DEVTEMP  6006 DOSCMD   6007 DOSCMD1  600D DOSCMD2 6016
BUFSET   601E BUFSET1  603A BUFSET2  6040 BUFREL  6046
BUFREL1  605B FOPEN    605E FOPEN1   6067 FOPEN2  6070
FOPEN3   6085 FOPEN4   6087 FOPEN5   609F FOPEN6  60A2
YYYY     60C0 SETDV6   60C4 SETDV7   60E7 FCLOSE  610A
FCLOSE2  6112 FCLOSE4  6118 FCLOSE6  614A GETCH   614B
PUTCH    6158 PUTSTR   616B PUTSTR1  6172 PUTSTR2 6186
MAIN     6187 MAIN1    61BB MESS1    61CA MESS2   61D9
FILENAME 61FD MESS3    6206

USER SYMBOLS   ( ALPHABETIC ORDER )
SYMBOL  VALUE
BUFREL   6046 BUFREL1 605B BUFSET   601E BUFSET1 603A
BUFSET2  6040 BUFVECT 00E1 CALLTK   2B1A CDKLUDGE 2C0D
DEVTEMP  6006 DOSBUFPT 2CE5 DOSCMD  6007 DOSCMD1 600D
OSCMD2  6016 DOSCMD2  6016 DOSINTRP 2A8A FCLOSE  610A
FCLOSE2  6112 FCLOSE4  6118 FCLOSE6  614A FILENAME 61FD
FILEVAR  2326 FINDFILE 2DA6 FOPEN   605E FOPEN1  6067
FOPEN2   6070 FOPEN3   6085 FOPEN4  6087 FOPEN5  609F
FOPEN6   60A2 GETCH    614B INPTAB  2301 IOCHR   2363
KLUDGDAT 0000 MAIN     6187 MAIN1   61BB MEMVECT 00FE
MESS1    61CA MESS2    61D9 MESS3   6206 OUTTAB  2311
PUTCH    6158 PUTSTR   616B PUTSTR1 6172 PUTSTR2 6186
SAVETK   2C37 SECTNM   265E SETDV6  60C4 SETDV7  60E7
SETTK    26BC STDERR   6005 STDIN   6003 STDOUT  6004
STROUT   2D73 VECT     00FC YYYY    60C0
ASSEMBLY COMPLETE, NO ERRORS
```

ALPHA and I.

By Frank Brown.

I had considered purchasing a printer for a long time before finally taking the big step, into COMP-SOFT where George made me an offer that I could not refuse. The next week I picked up my Alpha-80 printer. It is a step that I do not regret taking, the Alpha-80 was a steal and it has the features of a printer that cost $1000 last year.

Having bought my printer the next step was to fit it to my C1/4P hybrid. As I still have ROM Basic fitted and use Pico-Dos frequently I wanted to use the printer with ROM Basic programs. Device #1 was my choice as Device #4 is not accessable from Pico-Dos. The standard printer location is Device #4, I refer you to the KAOS newsletter Vol 3 No.9 page 7 if you use Device #4. COMP-SOFT supplied me with a Interface Board which provides a parallel printer connection from the ACIA socket while not interfering with the functions of the ACIA. Using this method, no additional software is required.

The assembly of the interface board is quite easy involving only six IC's and a few resistors, the most time consuming part was connecting the ribbon cable. The instructions in the printer manual on setting up were followed and the DIP switches, inside the printer, were left at factory settings. The Alpha-80 worked from the first time.

As C1/4P hybrids use a C4 DOS but are based on the C-1P the DOS must be changed to take note of the different location of the ACIA. The pokes below should be added to BEXEC* to correct the ACIA location in DOS.
Q=240:POKE9424,Q:POKE9432,Q:POKE9436,Q:POKE9464,Q:POKE9473,Q
C4 users with DABUG will have already made this change to allow the use of the cassette port. Next add two pokes to allow the printer to run at full speed.
POKE61440,3:POKE 61440,16

To use the printer from DOS, "LIST#1," or "PRINT#1," will print anything that's printable in BASIC. In Pico-Dos, do the two pokes above then SAVE (return), LIST (return). When printing is complete type LOAD (return) and then (space bar). If you have to change your DOS do make sure that ALL your disks have BEXEC* modified, the computer will lock up and you will loose the program if you attempt to write to a non-existant printer.

Now that I have a printer WP-6502 is in almost constant use. Below are some changes to WP-6502 set-up and printer set-up which I use. #C27#C69, this instruction to the printer sets it on emphasized mode. This mode, which is only slightly slower than normal print, produces good quality typewriter like characters. I have found that double strike mode produces characters that fill in and are difficult to read, it is also much slower.

To get a neater right hand margin I set the Default Text width to 80. Unfortunatly this caused stray commas to appear on the LH side of the page instead of at the end of the previous line. A line length of 79 causes a single full stop to be printed on the RH side of the page when the following page number (...x) is printed.

The printer has a large wire rack for the printed paper. I fitted this but found that it was more trouble than it was worth, the paper would catch on the rack and jam under the acoustic hood or the printer would go off line and the computer signal BREAK. Since the rack was removed there has been no trouble, besides it doubles the area that the printer occupies and I don't have the space.

I have found that to avoid the page follow-on (...2) being printed on the perforation after two pages have been printed the print head should be close to the perforation at the top of the page before printing starts.

I hope that the above is of use to all you new printer owners. Now what I need is a KAOS reader to tell me how to fix the graphics dump in OS65D3.3 so that I can dump the screen to my printer.

KAOS-W.A.

At our July meeting we had four computers, three new members and about ten 'old' members. We were able to some games demonstrated, including 'Snake Tails' from the May KAOS magazine (slightly modified to suit a C1P with Tasan video board). Other members indicated that they also had this game running with various modifications, such as speed, added to it.

Also being demonstrated was a Dick Smith modem hooked up to a C1P and talking to a Cyber (mainframe) at the W.A. University Computing Centre. There was also another sound generator being shown.

One of our new members, Lindsay Duus, indicated that he has just bought an 8" disk which he hopes to have hooked up to his C4P by our next meeting. If this is so he will be demonstrating it at our September meeting.

The date of our September meeting is Sunday 18/9/83 at 2pm, on the top floor of Guild House, 56 Kishorn St, Mt Pleasant.

At present we do not have a site for our November meeting so would members come with some ideas to the September meeting. (Ignore what I said inthe June KAOS about the September meeting place - I was out by two months - KAOS by name - KAOS by deed as well --.)
See you and your computers on the 18th.

Gerry Ligtermoet,

SYDNEY OSI GROUP

by Norman Bate

The bi-monthly meeting of the Sydney OSI Group took place on Sunday 24th July. During the day 16 people attended. One of features of the meeting was that all 7 computers were disk based.

David Samuel demonstrated his new Brother HR 15 daisy wheel printer and Tandy Plotter.

Eric Lindsay had several new programs from the U.S.A.

The next meeting is scheduled for Sunday 25th September at Lugarno. For further details please contact N. Bate

MODEM BOARD

Ed Richardson (OSUG) informs us that he has obtained more Racal modem boards complete with handbook. The boards are free and can be posted to anywhere in Australia, by certified mail, for $3. Ed suggests that if members get together and order a few for delivery to the one address they could save money on postage.

FOR SALE

SUPERBOARD Series II in case, 8K RAM, 8K ROM, P.S.G., Color, Four character sets, 5V 10A power supply plus lots of software and manuals. $400.00 O.N.O.
Peter

MPI B51 DISK DRIVE and manual, brand new (still in box)
Sell $280.00 Contact Rod

C4P, with 32K, 2MHz, one 8" D/S D/D floppy disk drive. Modified B&W TV, 2 Joysticks, much documentation.
$950.00 O.N.O. Contact Peter McLennan

Registered by Australia Post
Publication No. VBG4212

If undeliverable return to
KAOS, 10 Forbes St
Essendon, Victoria 3040

KAOSKAOS
K Postage K
A Paid A
O Essendon O
S 3040 S
KAOSKAOS
101179v3